Resulting pending claims for examination:

B1

A1

1. A method for generating a dump file, the method comprising:

   a. generating a dump file by gathering at least:

      i. thread information for at least one running thread,

      ii. context information for the thread,

      iii. callstack information for the thread,

      iv. process information for a process in which the thread is running, and

      v. information identifying a reason for generating the dump file; and

   b. storing the dump file to a storage medium.

2. The method as recited in Claim 1, further comprising determining when to generate the dump file.

3. The method as recited in Claim 1, wherein generating the dump file further includes gathering processor information about at least one processor.

4. The method as recited in Claim 2, wherein determining when to generate the dump file further includes determining that an exception has occurred.

B1
A1

5. The method as recited in Claim 1, wherein the dump file does not include data stored in global initialized memory.

6. The method as recited in Claim 1, wherein the dump file does not include data stored in uninitialized memory.

7. The method as recited in Claim 1, wherein the dump file does not include executable instructions used by a processor to execute a program.

8. The method as recited in Claim 1, wherein the dump file is a kernel minidump file associated with an operating system and the at least one running thread is the single thread which encountered an exception.

9. The method as recited in Claim 8, wherein the callstack information includes kernel stack information.

10. The method as recited in Claim 1, wherein the process information identifies a process that initiated the thread.

11. The method as recited in Claim 1, further comprising:

allocating a buffer space in memory during an initialization process, wherein the buffer space is suitable for storing the gathered information; and

B1
A1

reserving space on the storage medium suitable for writing the contents of the buffer space.

12. The method as recited in Claim 11, wherein generating the dump file further includes initially storing the thread information, the context information, the callstack information, the process information, and the information identifying the reason for generating the dump file to the buffer space, and then copying the dump file from the buffer space to the storage medium as a minidump file.

13. The method as recited in Claim 12, further comprising upon re-initialization, after having stored the minidump file to the storage medium, accessing the minidump file on the storage medium and using at least a portion of the minidump file to further understand an exception that was at least one reason for generating the minidump file.

14. The method as recited in Claim 1, wherein the dump file is a user minidump file associated with at least one non-operating system program.

15. The method as recited in Claim 1, wherein generating the dump file further includes gathering callstack information for all running threads.

16. The method as recited in Claim 15, wherein the callstack information includes a user callstack.

B1
AT

17. The method as recited in Claim 1, wherein generating the dump file further includes gathering processor context information for all running threads.

18. The method as recited in Claim 1, wherein generating the dump file further includes gathering a listing of loaded modules for a faulting application program.

19. The method as recited in Claim 1, wherein the dump file is a directory indexed file that uses relative virtual addresses (RVAs).

20. A computer-readable medium having computer-executable instructions for causing at least one processor to perform acts comprising:

gathering dump file information including at least thread information for at least one running thread, context information for the thread, callstack information for the thread, process information for the process in which the thread is running, and information identifying a reason for generating the dump file; and generating a dump file using the dump file information.

21. The computer-readable medium as recited in Claim 20, wherein generating the dump file further includes storing the dump file to a storage medium.

B1

A1

22. The computer-readable medium as recited in Claim 20, wherein gathering the dump file information further includes gathering processor information about at least one processor.

23. The computer-readable medium as recited in Claim 20, having further computer-executable instructions for causing the at least one processor to perform acts comprising determining when to generate the dump file.

24. The computer-readable medium as recited in Claim 20, wherein the dump file does not include data stored in global initialized memory.

25. The computer-readable medium as recited in Claim 20, wherein the dump file does not include data stored in uninitialized memory.

26. The computer-readable medium as recited Claim 24 wherein the dump file does not include executable instructions used by the at least one processor to execute a program.

27. The computer-readable medium as recited in Claim 20, wherein the dump file is a kernel minidump file associated with an operating system and the at least one running thread is the single thread which encountered an exception.

B1

A1

~~28. The computer-readable medium as recited in Claim 20, wherein the callstack information includes kernel stack information.~~

~~29. The computer-readable medium as recited in Claim 20, wherein the process information identifies a process that initiated the thread.~~

~~30. The computer-readable medium as recited in Claim 20, further comprising computer-executable instructions for causing the at least one processor to perform acts comprising:~~

~~allocating a buffer space in memory during an initialization process, wherein the buffer space is suitable for storing the dump file information; and~~

~~reserving space on a storage medium drive suitable for writing the contents of the buffer space.~~

~~31. The computer-readable medium as recited in Claim 30, wherein generating the dump file further includes initially storing the thread information, the context information, the callstack information, the process information, and the information identifying the reason for generating the dump file to the buffer space, and then copying the dump file from the buffer space to the storage medium as a minidump file.~~

B1
AT

~~32. The computer-readable medium as recited in Claim 31, further comprising computer-executable instructions for causing the at least one processor to perform acts comprising, upon re-initialization after having stored the minidump file to the storage medium, accessing the minidump file on the storage medium and using at least a portion of the minidump file to further understand an exception that was at least one reason for generating the minidump file.~~

~~33. The computer-readable medium as recited in Claim 20, wherein the dump file is a user minidump file associated with at least one non-operating system program.~~

~~34. The computer-readable medium as recited in Claim 20, wherein gathering the dump file information further includes gathering callstack information for all running threads.~~

~~35. The computer-readable medium as recited in Claim 34, wherein the callstack information includes a user callstack.~~

~~36. The computer-readable medium as recited in Claim 20, wherein gathering the dump file information further includes gathering processor context information for all running threads.~~

B1
AT

37. The computer-readable medium as recited in Claim 20, wherein gathering the dump file information further includes gathering a listing of all loaded modules for the faulting application program.

38. The computer-readable medium as recited in Claim 20, wherein the dump file is a directory indexed file that uses relative virtual addresses (RVAs).

39. An apparatus comprising;
memory;
a data storage drive configured to write data files to at least one data storage medium; and
at least one processor operatively coupled to the memory and the data storage drive and configured to:
a. generate a dump file by gathering in the memory at least:
i. thread information for at least one running thread,
ii. context information for the thread,
iii. callstack information for the thread,
iv. process information for the process in which the thread is running, and
v. information identifying a reason for generating the dump file; and
b. store the dump file to the storage medium.

B1

~~A1~~

~~40. The apparatus as recited in Claim 39, wherein the at least one processor is further configured to determine when to generate the dump file.~~

~~41. The apparatus as recited in Claim 39, wherein the at least one processor is further configured to gather processor information about the at least one processor and include the processor information in the dump file.~~

~~42. The apparatus as recited in Claim 40, wherein the at least one processor is further configured to determining when to generate the dump file based on an exception.~~

~~43. The apparatus as recited in Claim 39, wherein the dump file does not include data stored in global initialized memory.~~

~~44. The apparatus as recited in Claim 39, wherein the dump file does not include data stored in uninitialized memory.~~

~~45. The apparatus as recited Claim 39 wherein the dump file does not include executable instructions used by the at least one processor to execute a program.~~

~~46. The apparatus as recited in Claim 39, wherein the dump file is a kernel minidump file associated with an operating system and the at least one running thread is the single thread which encountered an exception.~~

B1
At

47. The apparatus as recited in Claim 39, wherein the callstack information includes kernel stack information.

48. The apparatus as recited in Claim 39, wherein the process information identifies a process that initiated the thread.

49. The apparatus as recited in Claim 39, wherein the at least one processor is further configured to:

allocate a buffer space in the memory during an initialization process; and

reserve space on the storage medium drive suitable for writing the contents of the buffer space.

50. The apparatus as recited in Claim 49, wherein the at least one processor is further configured to:

generate the dump file by initially storing the thread information, the context information, the callstack information, the process information, and the information identifying the reason for generating the dump file to the buffer space, and then copying the dump file from the buffer space to the storage medium as a minidump file.

51. The apparatus as recited in Claim 50, wherein the at least one processor is further configured to, upon re-initialization after having stored the minidump file to the storage medium, access the minidump file on the storage medium and use at least a portion of the minidump file to

~~further understand an exception that was at least one reason for generating the minidump file.~~

B1

~~52. The apparatus as recited in Claim 39, wherein the dump file is a user minidump file associated with at least one non-operating system program.~~

A1

~~53. The apparatus as recited in Claim 39, wherein the at least one processor is further configured to gather callstack information for all running threads as part of the dump file.~~

~~54. The apparatus as recited in Claim 53, wherein the callstack information includes a user callstack.~~

~~55. The apparatus as recited in Claim 39, wherein the at least one processor is configured to gather processor context information for all running threads as part of the dump file.~~

~~56. The apparatus as recited in Claim 39, wherein the at least one processor is configured to gather a listing of all loaded modules for a faulting application program as part of the dump file.~~

~~57. The apparatus as recited in Claim 39, wherein the dump file is a directory indexed file that uses relative virtual addresses (RVAs).~~

B1

A2

~~67. The method as recited in Claim 1, further comprising providing the dump file to at least one external device.~~

~~68. The method as recited in Claim 12, upon system re-initialization, transferring the dump file from the storage medium to at least one external device.~~

~~69. The method as recited in Claim 1, wherein generating the dump file further includes gathering a list of loaded modules.~~

~~70. The computer-readable medium as recited in Claim 20, having further computer-executable instructions for causing the at least one processor to perform acts comprising providing the dump file to at least one external device.~~

~~71. The computer-readable medium as recited in Claim 30, having further computer-executable instructions for causing the at least one processor to perform acts comprising, upon system re-initialization, transferring the dump file from the storage medium to at least one external device.~~

~~72. The computer-readable medium as recited in Claim 20, wherein gathering the dump file information further includes gathering a list of loaded modules.~~

B1

A2

~~73. The apparatus as recited in Claim 39, wherein the at least one processor is further configured to provide the dump file to at least one external device.~~

~~74. The apparatus as recited in Claim 49, wherein the at least one processor is further configured to, upon system re-initialization, transferring the dump file from the storage medium to at least one external device.~~

~~75. The apparatus as recited in Claim 39, wherein the at least one processor is further configured to gather a list of loaded modules as part of the dump file.~~

~~76. An application programming interface (API) method for use between a first process and a second process operatively configured on at least one processing unit in a computing device, the API method comprising:~~

~~a. issuing, by the first process, a write dump file call having a plurality of call parameters comprising a process handle, a process identifier, a handle to a file where dump file information is to be written, and a dump type identifier;~~

~~b. receiving, by the second process, the write dump file call and parsing the call to retrieve the parameters; and~~

~~c. issuing, by the first process, a write dump file call acknowledgment providing a true-false indication.~~

[illegible]

A2

~~77. An application programming interface (API) method for use between a first process and a second process operatively configured on at least one processing unit in a computing device, the API method comprising:~~

~~a. issuing, by the first process, a read dump file call having a plurality of call parameters comprising a header of a dump file and a data type identifier of data to read from a dump file;~~

~~b. receiving, by the second process, the read dump file call and parsing the call to retrieve the parameters; and~~

~~c. issuing, by the first process, a read dump file call acknowledgment providing a true-false indication and a plurality of call return parameters comprising a pointer to a beginning of a dump stream, and a stream size identifying the size of the dump stream.~~

[illegible]

Claim amendments shown below:

1. (Once Amended) A method for generating a dump file, the method comprising:

a. [determining when to generate a dump file; and

b.] generating a dump file by gathering at least:

i. thread information for at least one running thread,

ii. context information for the thread,

iii. callstack information for the thread,

iv. process information for a[the] process in which the thread is running, and

v. information identifying a reason for generating the dump file; and

b. storing the dump file to a storage medium.

2. (Once Amended)The method as recited in Claim 2, further comprising determining when to generate [wherein generating] the dump file [further includes storing the dump file to a storage medium].

5. (Once Amended) The method as recited in Claim 1[4], wherein the dump file does not [further] include data stored in global initialized [any significant portion of a dynamically allocated] memory.

6. (Once Amended) The method as recited in Claim 1[5] wherein the dump file does not include [any portion of a global initialized or] data stored in uninitialized memory.

7. (Once Amended) The method as recited in Claim 1[5] wherein the dump file does not include [any portion of the] executable instructions used by [the] a processor to execute [the] a program.

8. (Once Amended) The method as recited in Claim 1, wherein the dump file is a kernel minidump file associated with an operating system and the at least one running thread is the single thread which encountered [the] an exception.

9. (Once Amended) The method as recited in Claim 8[1], wherein the callstack information [is a] includes kernel stack information.

10. (Once Amended) The method as recited in Claim 1, wherein the process information identifies [the] a process that initiated the thread.

11. (Once Amended) The method as recited in Claim 1, further comprising:

[a.]allocating a buffer space in memory during an initialization process, wherein the buffer space is suitable for storing the gathered information; and [b.] reserving space on [a] the storage medium [drive] suitable for writing the contents of the buffer space.

12. (Once Amended) The method as recited in Claim 11, wherein[:

a.] generating the dump file further includes initially storing the thread information, the context information, the callstack information, the process information, and the information identifying the reason for generating the dump file to the buffer space, and then copying the dump file from the buffer space to the storage medium as a minidump file[; and

b. upon system re-initialization, transferring the dump file from the storage medium to at least one external computer].

13. (Once Amended) The method as recited in Claim 12, further comprising upon re-initialization, after having stored the minidump file to the storage medium, accessing the minidump file on the storage medium and using at least a portion of the minidump file to further understand an exception that was at least one reason for generating the minidump file.

16. (Once Amended) The method as recited in Claim 15[1], wherein the callstack information [is] includes a user callstack.

18. (Once Amended) The method as recited in Claim 1, wherein generating the dump file further includes gathering a listing of [all] loaded modules for [the] a faulting application program.

20. (Once Amended) A computer-readable medium having computer-executable instructions for causing at least one processor to perform[ing steps] acts comprising:

[a. determining when to generate a dump file; and

b. generating a dump file by] gathering dump file information including at least[:

i.] thread information for at least one running thread,

[ii.] context information for the thread,

[iii.] callstack information for the thread,

[iv.] process information for the process in which the thread is running, and

[v.] information identifying a reason for generating the dump file; and

generating a dump file using the dump file information.

22. (Once Amended) The computer-readable medium as recited in Claim 20, wherein [generating] gathering the dump file information further includes gathering processor information about at least one processor.

23. (Once Amended) The computer-readable medium as recited in Claim 20, [wherein] having further computer-executable instructions for

causing the at least one processor to perform acts comprising determining when to generate the dump file [further includes determining that an exception has occurred].

24. (Once Amended) The computer-readable medium as recited in Claim 20[23], wherein the dump file does not [further] include data stored in global initialized [any significant portion of a dynamically allocated] memory.

25. (Once Amended) The computer-readable medium as recited in Claim 20[24] wherein the dump file does not include [any portion of a global initialized or] data stored in uninitialized memory.

26. (Once Amended) The computer-readable medium as recited Claim 24 wherein the dump file does not include [any portion of the] executable instructions used by the at least one processor to execute [the] a program.

27. (Once Amended) The computer-readable medium as recited in Claim 20, wherein the dump file is a kernel minidump file associated with an operating system and the at least one running thread is the single thread which encountered [the] an exception.

28. (Once Amended) The computer-readable medium as recited in Claim 20, wherein the callstack information [is a] includes kernel stack information.

29. (Once Amended) The computer-readable medium as recited in Claim 20, wherein the process information identifies [the] a process that initiated the thread.

30. (Once Amended) The computer-readable medium as recited in Claim 20, further comprising computer-executable instructions for causing the at least one processor to perform[ing steps of] acts comprising:

allocating a buffer space in memory during an initialization process, wherein the buffer space is suitable for storing the dump file information; and

reserving space on a storage medium drive suitable for writing the contents of the buffer space.

31. (Once Amended) The computer-readable medium as recited in Claim 30, wherein generating the dump file further includes initially storing the thread information, the context information, the callstack information, the process information, and the information identifying the reason for generating the dump file to the buffer space, and then copying the dump file from the buffer space to the storage medium as a minidump file[; and

upon system re-initialization, transferring the dump file from the storage medium to at least one external different computer].

32. (Once Amended) The computer-readable medium as recited in Claim 31, further comprising computer-executable instructions for causing the at least one processor to perform[ing steps of] acts comprising, upon re-initialization after having stored the minidump file to the storage medium, accessing the minidump file on the storage medium and using at least a portion of the minidump file to further understand an exception that was at least one reason for generating the minidump file.

34. (Once Amended) The computer-readable medium as recited in Claim 20, wherein [generating the dump file] gathering the dump file information further includes gathering callstack information for all running threads.

35. (Once Amended) The computer-readable medium as recited in Claim 34[20], wherein the callstack information [is] includes a user callstack.

36. (Once Amended) The computer-readable medium as recited in Claim 20, wherein [generating the dump file] gathering the dump file information further includes gathering processor context information for all running threads.

37. (Once Amended) The computer-readable medium as recited in Claim 20, wherein [generating the dump file] gathering the dump file

information further includes gathering a listing of all loaded modules for the faulting application program.

39. (Once Amended) An [arrangement] apparatus comprising;

memory[,];

a data storage drive configured to write data files to at least one data storage medium[,]; and

at least one processor operatively coupled to the memory and the data storage drive and configured to:

a. [determine when to generate a dump file; and

b.] generate a dump file by gathering in the memory at least:

i. thread information for at least one running thread,

ii. context information for the thread,

iii. callstack information for the thread,

iv. process information for the process in which the thread is running, and

v. information identifying a reason for generating the dump file; and

b. store the dump file to the storage medium.

40. (Once Amended) The [arrangement] apparatus as recited in Claim 39, wherein the at least one processor is further configured to determine when to generate[ing] the dump file [further includes storing the dump file to a storage medium].

41. (Once Amended) The [arrangement] apparatus as recited in Claim 39, wherein the at least one processor is further configured to [generating the dump file further includes] gather[ing] processor information about the at least one processor and include the processor information in the dump file.

42. (Once Amended) The [arrangement] apparatus as recited in Claim 40[39], wherein the at least one processor is further configured to determining when to generate the dump file [further includes determining that] based on an exception [has occurred].

43. (Once Amended) The [arrangement] apparatus as recited in Claim 39[43], wherein the dump file does not [further] include data stored in global initialized [any significant portion of a dynamically allocated] memory.

44. (Once Amended) The [arrangement] apparatus as recited in Claim 39[43] wherein the dump file does not include [any portion of a global initialized or] data stored in uninitialized memory.

45. (Once Amended) The [arrangement] apparatus as recited Claim 39[43] wherein the dump file does not include [any portion of the] executable instructions used by the at least one processor to execute [the] a program.

46. (Once Amended) The [arrangement] apparatus as recited in Claim 39, wherein the dump file is a kernel minidump file associated with an operating system and the at least one running thread is the single thread which encountered [the] an exception.

47. (Once Amended) The [arrangement] apparatus as recited in Claim 39, wherein the callstack information [is a] includes kernel stack information.

48. (Once Amended) The [arrangement] apparatus as recited in Claim 39, wherein the process information identifies [the] a process that initiated the thread.

49. (Once Amended) The [arrangement] apparatus as recited in Claim 39, wherein the at least one processor is further configured to [further comprising computer-executable instructions for performing steps of]:

allocate[ing] a buffer space in the memory during an initialization process; and

reserve[ing] space on [a] the storage medium drive suitable for writing the contents of the buffer space.

50. (Once Amended) The [arrangement] apparatus as recited in Claim 49, wherein the at least one processor is further configured to:

generate[ing] the dump file [further includes] by initially storing the thread information, the context information, the callstack information, the process information, and the information identifying the reason for generating the dump file to the buffer space, and then copying the dump file from the buffer space to the storage medium as a minidump file[; and

upon system re-initialization, transferring the dump file from the storage medium to at least one external computer].

51. (Once Amended) The [arrangement] apparatus as recited in Claim 50, wherein the at least one processor is further [comprising computer-executable instructions for performing steps of] configured to, upon re-initialization after having stored the minidump file to the storage medium, access[ing] the minidump file on the storage medium and use[ing] at least a portion of the minidump file to further understand an exception that was at least one reason for generating the minidump file.

52. (Once Amended) The [arrangement] apparatus as recited in Claim 39, wherein the dump file is a user minidump file associated with at least one non-operating system program.

53. (Once Amended) The [arrangement] apparatus as recited in Claim 39, wherein the at least one processor is further configured to [generating the dump file further includes] gather[ing] callstack information for all running threads as part of the dump file.

54. (Once Amended) The [arrangement] apparatus as recited in Claim 53[39], wherein the callstack information [is] includes a user callstack.

55. (Once Amended) The [arrangement] apparatus as recited in Claim 39, wherein the at least one processor is configured to [generating the dump file further includes] gather[ing] processor context information for all running threads as part of the dump file.

56. (Once Amended) The [arrangement] apparatus as recited in Claim 39, wherein the at least one processor is configured to [generating the dump file further includes] gather[ing] a listing of all loaded modules for [the] a faulting application program as part of the dump file.

57. (Once Amended) The [arrangement] apparatus as recited in Claim 39, wherein the dump file is a directory indexed file that uses relative virtual addresses (RVAs).

--67. The method as recited in Claim 1, further comprising providing the dump file to at least one external device.

68. The method as recited in Claim 12, upon system re-initialization, transferring the dump file from the storage medium to at least one external device.

69. The method as recited in Claim 1, wherein generating the dump file further includes gathering a list of loaded modules.

70. The computer-readable medium as recited in Claim 20, having further computer-executable instructions for causing the at least one processor to perform acts comprising providing the dump file to at least one external device.

71. The computer-readable medium as recited in Claim 30, having further computer-executable instructions for causing the at least one processor to perform acts comprising, upon system re-initialization, transferring the dump file from the storage medium to at least one external device.

72. The computer-readable medium as recited in Claim 20, wherein gathering the dump file information further includes gathering a list of loaded modules.

73. The apparatus as recited in Claim 39, wherein the at least one processor is further configured to provide the dump file to at least one external device.

74. The apparatus as recited in Claim 49, wherein the at least one processor is further configured to, upon system re-initialization, transferring the dump file from the storage medium to at least one external device.

75. The apparatus as recited in Claim 39, wherein the at least one processor is further configured to gather a list of loaded modules as part of the dump file.

76. An application programming interface (API) method for use between a first process and a second process operatively configured on at least one processing unit in a computing device, the API method comprising:

a. issuing, by the first process, a write dump file call having a plurality of call parameters comprising a process handle, a process identifier, a handle to a file where dump file information is to be written, and a dump type identifier;
b. receiving, by the second process, the write dump file call and parsing the call to retrieve the parameters; and
c. issuing, by the first process, a write dump file call acknowledgment providing a true-false indication.

77. An application programming interface (API) method for use between a first process and a second process operatively configured on at least one processing unit in a computing device, the API method comprising:

a. issuing, by the first process, a read dump file call having a plurality of call parameters comprising a header of a dump file and a data type identifier of data to read from a dump file;

b. receiving, by the second process, the read dump file call and parsing the call to retrieve the parameters; and

c. issuing, by the first process, a read dump file call acknowledgment providing a true-false indication and a plurality of call return parameters comprising a pointer to a beginning of a dump stream, and a stream size identifying the size of the dump stream. --